Thought about RSA security and key management for a couple of seconds

Below is a structured solution for Tasks (b) and (c). Each task is broken down into:

1. **Overview/Report**: A detailed write-up discussing relevant theoretical points, practical considerations, and insights.
2. **Code Example**: Python snippets or modules that implement some aspects relevant to the task (e.g., demonstration code for factoring attempts, timing-attack simulations, or key management).

**Task (b): Analyze the Security of RSA Against Various Attacks**

**b.1. Overview/Report**

**1. Common Attacks on RSA**

1. **Factoring the Modulus (*n = p × q*)**
   * **Classical Factorization**: The security of RSA primarily relies on the computational difficulty of factoring large integers that are the product of two large primes. For sufficiently large key sizes (e.g., 2048 bits and above), classical factoring algorithms (e.g., trial division, Pollard’s Rho, Fermat’s factorization, ECM, GNFS) are not feasible with current technology in any reasonable timeframe.
   * **Quantum Attacks**: With the advent of quantum computing, Shor’s algorithm can factor large numbers in polynomial time. However, current quantum hardware is not yet at the scale needed to break 2048-bit RSA in practice.
2. **Timing Attacks**
   * Timing attacks exploit variations in decryption or signing time to recover private keys. An adversary measures the time it takes for the RSA device (or server) to process certain computations, which can leak information about the private exponent or prime factors.
   * **Countermeasures**: Implementing constant-time operations, blinding techniques, and other side-channel hardening measures can mitigate timing attacks.
3. **Chosen Ciphertext Attacks**
   * Attackers may craft special ciphertexts and feed them to a decryption oracle. If the oracle returns partial or error information, the attacker might glean secrets (e.g., via RSA padding oracles).
   * **Countermeasures**: Using secure padding schemes such as OAEP instead of PKCS#1 v1.5, and ensuring the decryption errors do not leak information (e.g., by returning only generic error messages).
4. **Implementation Vulnerabilities**
   * **Fault Attacks**: Hardware faults induced intentionally (e.g., glitching a chip) could skip or alter instructions in the RSA process, leaking private keys.
   * **Side-Channel Attacks**: Power analysis, electromagnetic emissions, and other physical side channels can also reveal private key bits.

**2. Factoring RSA Modulus (Naive Approach)**

Although classical factoring an RSA-2048 modulus is impractical, we can demonstrate a **naive** factoring approach for **small** RSA moduli as a concept demonstration. For very large moduli, this naive approach will be intractable, but it serves to illustrate the principle and confirm the importance of large key sizes.

**3. Timing Attack Simulation**

To illustrate a timing attack, we can measure how the RSA implementation’s decryption time might vary for different ciphertexts. In practice, we attempt to keep decryption constant-time. Our naive example might show micro-variations or artificially introduced variations. However, real timing attacks require very precise measurements and repeated queries.